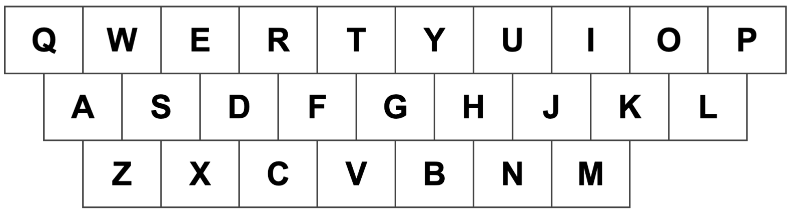
2021 LGE Code Jam: Online Round 1

Problem A. QWERTY Keyboard

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 sec
Memory limit:	512 MB



Albert wants to enter a string that consists of uppercase English letters ('A'-'Z') using a QWERTY keyboard (see the image above).

Albert isn't good at typing yet, and only wants to use his left index finger.

It takes 1 second to press a button to enter one character, and moving the finger from one button to its adjacent button takes 2 seconds (assume that the finger never leaves the keyboard while entering a string).

Each button has up to six adjacent buttons. For instance, moving the finger from S to A, W, E, D, X, or Z takes 2 seconds.

Albert starts by placing his index finger on the button for the first character in the string, and then measures time.

If "QWERTY" is to be entered, then it takes 16 seconds to enter it as follows:

- First, place the finger on button Q. (Time to do this is not measured.)
- Pressing and releasing the button (to enter Q) takes 1 second. (Total: 1 second)
- Moving to W takes 2 seconds. (Total: 3 seconds)
- Pressing and releasing the button (to enter W) takes 1 second. (Total: 4 seconds)
- Moving to E takes 2 seconds. (Total: 6 seconds)
- Pressing and releasing the button (to enter E) takes 1 second. (Total: 7 seconds)
- Moving to R takes 2 seconds. (Total: 9 seconds)
- Pressing and releasing the button (to enter R) takes 1 second. (Total: 10 seconds)
- Moving to T takes 2 seconds. (Total: 12 seconds)
- Pressing and releasing the button (to enter T) takes 1 second. (Total: 13 seconds)
- Moving to Y takes 2 seconds. (Total: 15 seconds)
- Pressing and releasing the button (to enter Y) takes 1 second. (Total: 16 seconds)

It takes 9 seconds to enter "LOM".

- Pressing and releasing the button (to enter L) takes 1 second. (Total: 1 second)
- Moving to O takes 2 seconds. (Total: 3 seconds)

- Pressing and releasing the button (to enter O) takes 1 second. (Total: 4 seconds)
- Moving to M takes 4 seconds (O -¿ K -¿ M is the fastest way to move). (Total: 8 seconds)
- Pressing and releasing the button (to enter M) takes 1 second. (Total: 9 seconds)

Given a string consisting only of uppercase English alphabets ('A'-'Z'), compute the fastest time (in seconds) needed for Albert to enter the string.

Input

The first line of input will contain T, the number of test cases.

For each test case, a single will contain a string (with no whitespaces) consisting of uppercase English alphabets.

Output

For each test case, output the least amount of time (in seconds) that Albert needs to enter the given string.

Constraints

- $1 \leq T \leq 10$
- $2 \leq$ Length of an input string ≤ 100
- Each string only contains uppercase English alphabets ('A'-'Z')

Examples

standard input	standard output
5 QWERTY LOM FFGGFF VGTRDCF MPML	16 9 10 19 20

Case 1: Explained in the problem statement.

Case 2: Explained in the problem statement.

Case 3: It takes 2 seconds to enter F twice, 2 seconds to move from F to G, 2 seconds to enter G twice, 2 seconds to move from G to F, and 2 seconds to enter F twice. Overall, it takes 10 seconds.

Case 4: It takes 1 second to enter each character (7 seconds in total), and it takes 2 seconds each to move between adjacent buttons (12 seconds in total). Overall, it takes 19 seconds.

Case 5: It takes 6 seconds each to move from M to P or to move from P to M, and it takes 4 seconds to move from M to L. Including the time needed to press the buttons, it takes 20 seconds, overall.

Problem B. Number Card Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 sec
Memory limit:	512 MB

Albert has n cards with digits written on them.

Each card has exactly one of the digits between 0 and 9 written on it, and cards with a 6 or a 9 cannot be distinguished (you can rotate the card to turn 6 to 9 or turn 9 to 6).

Albert wants to make two numbers using all of his cards such that the product of the two numbers would be maximized.

He must use all of his n cards, and each number must use 1 card (and at most $n - 1$ cards). Cards with a 6 or a 9 can be rotated arbitrarily.

For instance, suppose that $n = 8$ and Alber's cards are given by [2, 0, 2, 0, 2, 0, 2, 1].

Then, Albert can make two numbers "2200" and "2210", whose product is 4862000.

Or, he can make "2020" and "2021", whose product is 4082420.

In this example, the maximum product Albert can achieve is 4862000.

Given n cards, find the maximum product Albert can achieve.

Input

The first line will contain the number of test cases, T.

In each of the next T lines, Albert's cards will be described as a string consisting of digits ('0'-'9') without space.

Output

For each test case, output the maximum product Albert can achieve.

Constraints

- $1 \le T \le 10$
- $2 \le n \le 18$
- You may assume that the answer is at most 10^{18} for all test cases.

Examples

standard input	standard output
5 90000 66 102030 20202021 999999999999999999	0 81 63000 4862000 999999998000000001

Case 1: No matter how you make two numbers, one of the must be 0.

Case 2: Albert can rotate the two cards to make 9*9 = 81.

Case 3: 21000 * 3 = 63000.

Case 4: Explained in the problem statement.

Case 5: 999999999 * 999999999 = 999999998000000001.

Problem C. Bye 2020 Hello 2021

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1.5 sec
Memory limit:	512 MB

Albert wants to solve a problem as 2020 is over and 2021 is here.

For any positive integer, if the first four digits are "2020" and the last four digits are "2021", then the number is said to be **meaningful**.

For instance, 202021 and 20202021 are meaningful, but 2020021 or 2020221 are not.

Given n integers $A[1], A[2], \dots, A[n]$, Albert wants to calculate the number of pairs of integers whose sum is "meaningful."

That is, he wants to compute the number of pairs (i, j) with $1 \le i < j \le n$ such that $A[i] + A[j]$ is meaningful.

For instance, suppose that there are $n = 4$ integers, $A = [101010, 101010, 101011, 101011]$.

In this case, $A[1]+A[3] = A[1]+A[4] = A[2]+A[3] = A[2]+A[4] = 202021$, and thus there are four pairs of interest: $((1,3),(1,4),(2,3),(2,4))$.

Given n integers, compute the number of pairs of integers whose sum is meaningful.

Input

The first line will contain the number of test cases, T.

Each test case will be described in two lines.

The first line will contain n, the number of integers to be given.

The second line will contain n integers (separated by a whitespace).

Output

For each test case, output the number of pairs of numbers whose sum is meaningful.

Constraints

- $1 \le T \le 10$
- $2 \le n \le 100,000$
- $-2^{28} \le A[i] \le 2^{28}$

Examples

standard input	standard output

3
4
101010 101010 101011 101011
5
100000 100000 100000 101011 101011
4
202021 0 1 202020
	4
	0
	2

Case 1: Explained in the problem statement.

Case 2: No pair of integers adds to a meaningful number.

Case 3: There exist two pairs: (1, 2) and (3, 4) as 202021 + 0 and 1 + 202020 are both meaningful.

Problem D. Socially Distancing

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 sec
Memory limit:	512 MB

Albert needs to help setup a Hackathon event, adhering to social distancing rules.

There will be a long corridor (hallway) along which a booth with a desk, chair, and monitor will be installed.

Each booth can accommodate one team.

s teams will participate in the event, and there are n locations (electronic outlets) at which a booth can be set up.

Let $x[1], x[2], \ldots, x[n]$ be locations of the outlets ($x[i]$ is the distance from the corridor entrance).

That is, the i-th outlet is located at $x[i]$ meters from the entrance.

Albert needs to select s locations out of n outlets while maximizing the distance (call it D) between two closest outlets.

For instance, suppose $n = 3, s = 3$, and $x = [10, 100, 200]$. In this case, because $n = s$, each booth must be set up at each location. The closest distance between booths is (100 - 10) = 90.

In another example, suppose $n = 6, s = 4$, and $x = [11, 19, 24, 26, 29, 30]$. In this case, choosing four outlets at $x[1] = 11, x[2] = 19, x[3] = 24$, and $x[4] = 29$ ensures that the closest distance between booths is 5. Or, Albert can also choose $x[1] = 11, x[2] = 19, x[3] = 24$, and $x[4] = 30$ with the same closest distance. In this example, it is impossible to set up 4 booths with closest distance greater than or equal to 6.

Given n, s, and $x[1], ..., x[n]$ as input, find the maximum D that Albert can achieve.

Input

The first line will contain the number of test cases, T.

Each test case will be described in two lines.

The first line will contain two integers, n and s, separated by a whitespace.

The next line will contain n integers separated by a whitespace, describing n locations at which outlets are installed.

Output

For each test case, output the maximum D that Albert can achieve.

Constraints

- $1 \le T \le 10$
- $2 \le s \le n \le 200,000$
- $1 \le x[i] \le 1,000,000,000$
- You may assume that $x[i]$'s are distinct.

Examples

standard input	standard output

3
3 3
10 100 200
7 3
28 11 17 19 21 22 23
6 4
11 19 24 26 29 30
	90
	8
	5

Case 1: Explained in the problem statement.

Case 2: Choosing [11, 19, 28] yields $D = 8$.

Case 3: Explained in the problem statement.

Problem E. Good Arrays

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 sec
Memory limit:	256 MB

Albert is working on a fun problem.

If an array A contains n+1 numbers such that it contains each integer from 1 to n-1 exactly once and it contains n twice, then A is a **good array**.

There are exactly $(n+1)!/2$ good arrays of length n+1.

If A is a good array of length $n+1$, $Score(A)$ is defined as follows: The number of pairs (i, j) where $1 \leq i < j \leq n+1$ and $A[i] < A[j]$.

If $A = [1, 2, 3, ..., n-1, n, n]$, then $Score(A) = (n+2)(n-1)/2$ (which is maximum)

and if $A = [n, n, n-1, n-2, ..., 2, 1]$, then $Score(A) = 0$ (which is minimum).

Given n and two integers a and b, Albert wants to count the number of good arrays A of length $n+1$ such that $a \leq Score(A) \leq b$.

For instance, suppose that $n = 3, a = 2$, and $b = 3$. Out of 12 good arrays of length $n+1$, there exist 6 good arrays such that $a \leq Score(A) \leq b$.

- $A = [1, 2, 3, 3]$: $Score(A) = 5$
- $A = [1, 3, 2, 3]$: $Score(A) = 4$
- $A = [1, 3, 3, 2]$: $Score(A) = 3$ (OK)
- $A = [2, 1, 3, 3]$: $Score(A) = 4$
- $A = [2, 3, 1, 3]$: $Score(A) = 3$ (OK)
- $A = [2, 3, 3, 1]$: $Score(A) = 2$ (OK)
- $A = [3, 1, 2, 3]$: $Score(A) = 3$ (OK)
- $A = [3, 1, 3, 2]$: $Score(A) = 2$ (OK)
- $A = [3, 2, 1, 3]$: $Score(A) = 2$ (OK)
- $A = [3, 2, 3, 1]$: $Score(A) = 1$
- $A = [3, 3, 1, 2]$: $Score(A) = 1$
- $A = [3, 3, 2, 1]$: $Score(A) = 0$

Given n, a, and b, compute the number of good arrays A of length $n+1$ such that $a \leq Score(A) \leq b$ is satisfied.

Input

The first line will contain the number of test cases, T.

Each test case will be described in a single line that will contain n, a, and b, separated by a whitespace.

Output

For each test case, output the answer in a single line.

Since the answer can be very large, output the answer modulo $10^9 + 7$ (= 1,000,000,007).

Constraints

- $1 \le T \le 10,000$
- $2 \le n \le 1,000$
- $0 \le a \le b \le 1,000$

Examples

standard input	standard output
5	6
3 2 3	4
4 0 1	22
4 4 5	123924
8 10 20	113510379
12 0 77	

Case 1: Explained in the problem statement.

Case 2: These four arrays meet the requirements: [4, 4, 3, 2, 1], [4, 4, 3, 1, 2], [4, 4, 2, 3, 1], [4, 3, 4, 2, 1]

Case 3: No explanation is provided.

Case 4: No explanation is provided.

Case 5: All good arrays of length 13 satisfy the requirements, so the answer is 3113510400. However, you must output the answer modulo $10^9 + 7$, which is 113510379.

Problem F. XOR Data Structures

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 sec
Memory limit:	512 MB

Albert is interested in binary XOR operations and data structures.

Albert wants to create a data structure that can efficiently process XOR operations.

At the beginning, the data structure must be initialized with an input array $S0$ that contains n integers.

Then, the data structure must process q operations and output correct answers.

There are five types of operations, and some operations may modify the numbers stored within the data structure (hence, subsequent operations' results will be affected by a previous operation).

1. find_min(v): If S is the current set of numbers stored in the data structure, find an element s in S that minimizes (v XOR s), and output (v XOR s). S remains unchanged.
2. find_max(v): If S is the current set of numbers stored in the data structure, find an element s in S that maximizes (v XOR s), and output (v XOR s). S remains unchanged.
3. add(v): Add v to the current set S of numbers stored in the data structure. Then, output the number of unique numbers in S.
4. remove_min(): Output the smallest number stored in S of the data structure, and remove it. If there are multiple smallest numbers in S, remove them all.
5. remove_max(): Output the largest number stored in S of the data structure, and remove it. If there are multiple largest numbers in S, remove them all.

For instance, let $S0 = 1, 1, 3, 3$ (which initializes "S" of the data structure), and we process the following $q = 7$ operations.

1. find_min(2): Because (1 XOR 2) $= 3$ and (2 XOR 3) $= 1$, it must output 1. Afterwards, S remains unchanged, so $S = 1, 1, 3, 3$.
2. find_max(2): Because (1 XOR 2) $= 3$ and (2 XOR 3) $= 1$, it must output 3. Afterwards, S remains unchanged, so $S = 1, 1, 3, 3$.
3. add(2): By adding a new number 2 to S, now S becomes $S = 1, 1, 2, 3, 3$. There are three unique numbers in S, so it must output 3.
4. remove_min(): It must output 1, the smallest number in S, and remove all 1's. Afterwards, $S = 2, 3, 3$.
5. remove_max(): It must output 3, the largest number in S, and remove all 3's. Afterwards, $S = 2$.
6. find_min(2): Because (2 XOR 2) $= 0$, it must output 0.
7. find_max(2): Because (2 XOR 2) $= 0$, it must output 0.

In the example above, the correct output (in order of the given operations) is $[1, 3, 3, 1, 3, 0, 0]$.

Given n, $S0$, q, and details of the q operations, output q numbers (outputs obtained by applying the given operations).

Input

The first line will contain the number of test cases, T.

The first line of each test case will contain two integers, n and q, separated by a whitespace.

The second line of each test case will contain n integers, separated by a whitespace.

Each of the following q lines will describe one operation.

Each line will contain 1 or 2 integers (separated by a whitespace) where the first integer describes the type of an operation (one of $1, 2, 3, 4, 5$).

As described in the problem statement, 1 refers to find_min, 2 refers to find_max, 3 refers to add, 4 refers to remove_min, and 5 refers remove_max.

For operations of type 1, 2, or 3, the same line will contain the second integers, "v".

Output

For each operation given in the input, output the correct answer in a single line.

Constraints

- $1 \leq T \leq 10$
- $1 \leq n, q \leq 50,000$
- For every number v (given along with operations), assume $0 \leq v < 2^{25}$
- For every number s in $S0$, assume $0 \leq s < 2^{25}$
- For every operation, you may assume that S is non-empty both before and after processing the operation.

Examples

standard input	standard output
3	1
4 7	3
1 1 3 3	3
1 2	1
2 2	3
3 2	0
4	0
5	0
1 2	0
2 2	15
10 11	15
1 3 5 7 9 2 4 6 8 10	10
1 6	1
1 8	10
2 6	0
2 8	18
3 10	11
4	25
5	3
1 2	0
1 17	23
2 2	25
2 17	6
5 11	2
2 5 8 13 17	17
1 6	7
1 8	20
2 6	15
2 8	28
3 10	
4	
5	
1 2	
1 17	
2 2	
2 17	

Case 1: Explained in the problem statement.

Case 2: No explanation is provided.

Case 3:

Before processing any operations: $S = 2, 5, 8, 13, 17$

1. find_min(6): (6 XOR 5) = 3
2. find_min(8): (8 XOR 8) = 0
3. find_max(6): (6 XOR 17) = 23
4. find_max(8): (8 XOR 17) = 25
5. add(10): S = 2, 5, 8, 10, 13, 17 (6 unique numbers)

6. remove_min(): S = 5, 8, 10, 13, 17 (2 has been removed)
7. remove_max(): S = 5, 8, 10, 13 (17 has been removed)
8. find_min(2): (2 XOR 5) = 7
9. find_min(17): (17 XOR 5) = 20
10. find_max(2): (2 XOR 13) = 15
11. find_max(17): (17 XOR 13) = 28